AD-A231 558

NAVAL POSTGRADUATE SCHOOL

Monterey, California

DTIC
ELECTE
FEB 04 1991
S E D

THESIS

A TOOLKIT FOR DESIGNING USER INTERFACES

by

Susan Lynn Dunlap

March 1990

Thesis Advisor: Michael J. Zyda

Approved for public release; distribution is unlimited

91 2 01 054

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable) Code 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

A TOOLKIT FOR DESIGNING USER INTERFACES

12. PERSONAL AUTHOR(S)

Dunlap, Susan L.

13a. TYPE OF REPORT	13b TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM ____ TO ____	1990, March	78

16. SUPPLEMENTARY NOTATION

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

Interface; Graphics

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Current methods of developing user interfaces for IRIS workstation application programs are inefficient. In order to help speed the development of complex graphics programs, IRIS workstation users need a toolkit that will assist in the design and implementation of user interfaces for graphics programs. This project presents the preliminary work on an interface generator for the Silicon Graphics, Inc. IRIS workstation. The NPS Interface Builder (NPS IB) is designed to speed the creation of application programs by allowing a user to define an interface graphically rather than by writing "C" code. The program provides on-screen editing, facilitated by a number of program features. NPS IB can be used to develop the basic framework of a graphics program, or can be user to enhance the capabilities of an already existing graphics application.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. Michael J. Zyda	22b. TELEPHONE (Include Area Code) (408) 646-2305 22c. OFFICE SYMBOL Code 52Zk

DD FORM 1473, 84 MAR   83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24.

UNCLASSIFIED

Approved for public release; distribution is unlimited

A Toolkit for Designing User Interfaces

by

Susan Lynn Dunlap
Lieutenant, United States Navy
B.A., Northwestern University, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1990

Author: *Susan L. Dunlap*
Susan L. Dunlap

Approved by: *Michael J. Zyda*
Michael J. Zyda, Thesis Advisor

Leigh W. Bradbury
Leigh W. Bradbury, Second Reader

Robert B. McGhee
Robert B. McGhee, Chairman,
Department of Computer Science

ABSTRACT

Current methods of developing user interfaces for IRIS workstation application programs are inefficient. In order to help speed the development of complex graphics programs, IRIS workstation users need a toolkit that will assist in the design and implementation of user interfaces for graphics programs. This project presents the preliminary work on an interface generator for the Silicon Graphics, Inc. IRIS workstation. The NPS Interface Builder (NPS IB) is designed to speed the creation of application programs by allowing a user to define an interface graphically rather than by writing "C" code. The program provides on-screen editing, facilitated by a number of program features. NPS IB can be user to develop the basic framework of a graphics program, or can be user to enhance the capabilities of an already existing graphics application.

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

DTIC COPY INSPECTED

TABLE OF CONTENTS

I. THE NEED FOR AN INTERFACE SYSTEM ---------------- 1

A. CURRENT INTERFACE DESIGN METHOD ------------- 1

B. DEVELOPMENT OF INTERFACE BUILDERS AND TOOLKITS ------------------------------------ 2

1. Dialogue Management System (DMS) -------- 4

2. NASA Ames Panel Package ----------------- 5

3. Slider and Button Package --------------- 6

4. NPS Interface Builder ------------------- 6

C. NAVAL POSTGRADUATE SCHOOL GRAPHICS AND VIDEO LABORATORY ---------------------------- 7

D. SUMMARY OF THE CHAPTERS -------------------- 8

II. NPS INTERFACE BUILDER PROGRAM DESCRIPTION ------- 9

A. SYSTEM OVERVIEW ---------------------------- 9

III. THE NPS INTERFACE BUILDER PROGRAM --------------- 16

A. DESIGN ISSUES ------------------------------ 16

1. NPS Interface Builder User Interface ---- 16

2. NPS Interface Builder Program Design ---- 18

B. DATA STRUCTURES ---------------------------- 20

1. Saving Graphics Information ------------- 20

2. The Linked List Structure --------------- 24

C. SYSTEM STARTUP ----------------------------- 24

D. MAIN DRIVER ROUTINE ------------------------ 25

E. REDRAW/RESIZE CODE ------------------------- 27

F. DRAWING ALGORITHMS ------------------------- 31

1. Control Palette Window Drawings --------- 31

2. Design Window Drawing ------------------- 32

G. EDIT DESIGN MODE ---------------------------- 37

1. IRIS Picking Mode ----------------------- 37

2. Control Selection ----------------------- 38

3. Creating Record Structures for Saving Graphics Information ------------------- 38

H. MOVE CONTROL MECHANISM ---------------------- 39

I. RUN CONTROL MECHANISM ----------------------- 43

IV. NPS INTERFACE BUILDER MESSAGE BUFFER ------------ 45

A. PURPOSE OF THE MESSAGE BUFFER --------------- 45

B. OPERATION OF MESSAGE BUFFER WINDOW ---------- 45

V. NPS INTERFACE BUILDER OUTPUT -------------------- 51

A. SAVING INFORMATION TO OUTPUT FILES ---------- 51

B. SUPPORT ROUTINES FOR THE "C" LANGUAGE OUTPUT FILE ---------------------------------- 54

C. THE ASCII OUTPUT FILE AND THE RE-EDIT FEATURE -------------------------------------- 56

VI. CONCLUSIONS, LIMITATIONS, AND FUTURE WORK ------- 59

A. CONCLUSIONS AND LIMITATIONS ----------------- 60

1. Controls -------------------------------- 60

2. Color ----------------------------------- 61

3. Icon Labelling -------------------------- 62

B. FUTURE WORK --------------------------------- 62

1. Expanded Inventory of Controls ---------- 63

2. Adding Controls to NPS Interface Builder -------------------------------- 63

3. Interactive Size, Shape, and Color Adjustment ---------------------------- 64

4. Improving Interface of NPS Interface Builder ---------------------------------- 64

LIST OF REFERENCES -------------------------------------- 66

INITIAL DISTRIBUTION LIST ------------------------------ 67

LIST OF FIGURES

2.1 NPS Interface Builder Program Layout ------------ 10

2.2 Sample NPS Interface Builder Window Rearrangement ---------------------------------- 11

2.3 Design Window with Fixed Control ---------------- 12

2.4 Sample Interface Created with NPS Interface Builder -- 13

2.5 NPS Interface Builder ASCII File Output Listing -- 14

2.6 NPS Interface Builder "C" Language File Output Listing ---------------------------------- 15

3.1 Program Layout Showing Included Files ----------- 19

3.2 Toggle Box Record Structure --------------------- 21

3.3 Dial Record Structure --------------------------- 22

3.4 Vertical Slider Record Structure ---------------- 23

3.5 NPS Interface Builder System Menu --------------- 26

3.6 Code to Set Global Coordinates for Drawing Control Icons ---------------------------------- 29

3.7 Program Code for Drawing Floating Control ------- 33

3.8 Program Code for Drawing Fixed Controls --------- 35

3.9 Toggle Box Drawing Routine ---------------------- 36

3.10 Record Creation and List Maintenance for a Toggle Box -------------------------------------- 40

3.11 Searching Linked Lists for a Control ------------ 41

3.12 Code for Reconnecting a Moved Control Structure in the Linked List ------------------------------ 43

3.13 List Traversal and Toggle Box Operation --------- 44

4.1 Structure that Defines a Line of Text ----------- 46

4.2	Structure that Defines Message Buffer Window ----	47
4.3	Code for Constructing Ring Buffer ---------------	48
4.4	Adding a Line of Text to the Ring Buffer --------	50
5.1	Code for Writing from Linked Lists to Output Files ---	53
5.2	Code for Entering Information into the Toggle Box Array ---------------------------------------	55

ACKNOWLEDGMENTS

I wish to thank LT Laura J. White, USN, and Rosemary Lande for their work on the graphics class program that was the initial version of NPS IB. Although that work was abandoned in favor of another approach, their help on the project was invaluable in that it gave me a basic appreciation for graphics and a better understanding of the "C" programming language.

Thanks to LT Michael DeHaemer, USN, and LCDR Richard M. Prevatt, USN, for their assistance when I got stuck.

Thanks to Dr. Michael J. Zyda who persevered as my thesis advisor.

I. NEED FOR AN INTERFACE SYSTEM

This project is the preliminary work on an interface generator for the Silicon Graphics, Inc. IRIS workstation. The result of this project is the NPS Interface Builder (NPS IB), which is designed to speed the creation of application programs by allowing a user to define an interface graphically rather than writing "C" code. The development of the NPS IB represents a unique departure from previous methods of user interface development. This is because the NPS IB is capable of converting a user-specified picture into usable "C" code.

A. CURRENT INTERFACE DESIGN METHOD

Design of a user interface represents a considerable time investment for programmers. For some programmers, user interface concerns can mean an unwelcome diversion of time and energy away from an application program. Besides considering how the user interface should look, the program developer has to produce the actual implementation by writing original "C" code routines to draw the interface and produce the interaction between interface and program. This methodology is grossly inefficient, since it means that each user wishing to develop a graphics program for the IRIS workstation must essentially reinvent the wheel by coding his own interface. In addition to being inefficient, this

method of programming results in a wide disparity in the appearance of user interfaces across graphics programming. To an end-user, each graphics program presents a new and different set of interface challenges to master.

Certain types of interface controls seem to remain constant throughout graphics programs. Sliders are used to rotate, scale, or translate objects. Checkboxes are used to indicate that some function should be performed, such as printing data or verifying information. Dials can be used to represent mechanical devices such as a shipboard rudder angle indicator. Although the uses of the controls can vary from program to program, the basic operating mechanisms remain the same. This is what makes it possible to incorporate these controls into a standard library of controls. With the addition of routines to specify the placement and checkout of the controls, an interface builder for the IRIS workstation is born. The use of an interface builder can result in a considerable time-savings for the application programmer, and its use of a standard set of controls has the potential to relieve the problems caused for end-users by the large numbers of disparate interfaces that currently exist in graphics programming.

B. DEVELOPMENT OF INTERFACE BUILDERS AND TOOLKITS

Increase in the memory capacity of modern computers has been accompanied by a corresponding increase in the sophistication and size of application programs. As the

capabilities of the computer have expanded, end-users have demanded software that best utilizes the powers and abilities of their machines. This has resulted in an explosion of large, complex programs with highly sophisticated user interfaces. Often these programs are of such size and complexity that they are beyond the capabilities of just one person to write and debug.

Computer-aided software engineering is intended to reduce the time needed to create a software application, while at the same time improving the quality of the software produced. Reference 1 notes that the definition of computer-aided software engineering includes the use of tools which provide leverage at any point in the software development lifecycle. Interface generators, like the NPS IB, provide leverage in both the design and implementation phases of the lifecycle by allowing a designer to visually develop the interface and then automatically create the code for it.

Interface generators are also valuable tools for prototyping. Prototyping permits a software developer to quickly build an executable system which may not be complete. This system can then be demonstrated to users and subsequently can be easily modified if required. Prototyping saves time and money by permitting modifications to a system before the implementation phase of the software

lifecycle. An interface generator such as the NPS IB can provide prototyping capability.

Interface generators are available commercially for nearly every computing machine on the market. Such systems include Interface Builder, which permits users to build interfaces for application programs on the Next computer [Ref. 2], and C-scape [Ref. 3], which permits users to create custom interfaces for personal computers. In addition, several interface generators and toolkits have been developed specifically for the IRIS workstation family of computers. These include the following packages.

1. Dialogue Management System (DMS)

Reference 4 reports on an experimental research project at Virginia Tech called the Dialogue Management System. Developed for the Silicon Graphics IRIS 2400 workstation, DMS is described as a comprehensive system for interface management. The system is comprised of three components, including a dialogue component which provides for interaction between an end-user and the application program, a computational component, which contains all semantic processing algorithms, and a global control component, which manages sequencing between the dialogue and computational components. DMS provides for interactive development of all three components, including a tool for direct manipulation of graphical objects when developing an interface. The developers of DMS report that this

tool-based approach is much faster than conventional source coding. In fact, they say, the use of DMS for implementing an interface provides a nearly four-to-one improvement in speed over the use of programming for developing the same interface.

2. NASA Ames Panel Package

The NASA Ames Panel Package was developed by David Tristram at NASA Ames Research Center [Ref. 5]. The package is public-domain software, and provides a user-interface toolkit for the Silicon Graphics family of workstations. The Panel Package provides a panel library consisting of actuators, which are controls such as dials or sliders, and panels, which contain groups of actuators. Demonstration programs created using the panel package show that the package is a very powerful and professional-looking toolkit. Unfortunately, use of the panel package relies upon an understanding of package features, as well as a good understanding of the "C" programming language. Reference 3, which is the documentation manual for the panel package, can help a user get started. The time overhead required to learn the system does not offset the usefulness of the package. But the speed of producing an interface must inherently be slower than methods such as DMS since the user must deal directly with raw code.

3. <u>Slider and Button Package</u>

The Slider and Button (S & B) package is a tool for providing windows of sliders and buttons only on the Silicon `raphics IRIS 4D workstations. The S & B package consists of support routines which open a graphics window, draw the buttons and sliders, and provide the interaction mechanism between the controls and the end-user. The S & B package is quick and easy to use. The application programmer does not have to understand detailed operation of the package to use it. In this, the S & B package can realize a considerable time-savings for the programmer.

4. <u>NPS Interface Builder</u>

The NPS IB is the product of this study. It was designed to help speed the creation of application programs by allowing a user to define an interface graphically rather than writing "C" code. The program allows a designer to select interface controls from a standard set of controls, and to interactively specify the placement of those controls in a graphics window. The NPS IB produces as output the skeleton of a graphics program, written in "C" code, with designer-specified controls in place. The application programmer then builds his program upon this interface foundation. The NPS IB program can save time and effort for an application programmer, since it generates "C" code from a programmer-specified picture.

C. NAVAL POSTGRADUATE SCHOOL GRAPHICS AND VIDEO LABORATORY

The Naval Postgraduate School's Graphics and Video Laboratory has been developed for use by students in beginning graphics courses and for those students desiring to pursue master's thesis topics in graphics or graphics-related areas. The equipment currently in the lab consists of two Silicon Graphics IRIS-4D/120GTX workstations and one IRIS-4D/70GT workstation. These machines support project work from graphics courses as well as research work for master's thesis topics. The NPS IB program can benefit graphics course students and thesis research students equally as well. For the graphics student, NPS IB can produce the framework for a first graphics program--a framework upon which the student can build a graphics project as his learning and expertise grow. A graphics thesis student might use the NPS IB program to build supplemental command and control windows for a thesis program.

Thesis students using the Graphics and Video Laboratory have produced many visual simulators for various Department of Defense interests. These projects have included the Fiber Optic Guided Missile (FOGM), which modeled the flight of a missile over the terrain of Fort Hunter Liggett, California, and the Moving Platform Simulator, which featured the display of moving vehicles over terrain at Fort Hunter Liggett. These are just two examples of thesis

programs which could have benefited from use of the NPS IB program.

D. SUMMARY OF THE CHAPTERS

The remainder of this study discusses design and implementation issues considered in creating NPS IB. Chapter II provides an overview of the NPS IB, from its operating environment to program features. Chapter III discusses program specifics including graphics support and implementation considerations. Chapter IV discusses operation of the program message buffer. Chapter V focuses on the format of NPS IB output, along with the routines that produce the output. Chapter VI contains conclusions, limitations, and future directions.

II. NPS INTERFACF BUILDER PROGRAM DESCRIPTION

A. SYSTEM OVERVIEW

The NPS IB is a menu-driven program consisting of a design window, a control palette window, and a message and instruction window, shown in Figure 2.1. The design window represents the application programmer's easel, of sorts, upon which the layout of the user interface is designed. The control palette window contains representations of the user interface controls from which the application programmer can choose. The message and instruction window provides messages that confirm menu choices and that offer instructional help designed to guide the programmer through use of the program. The design window offers a resize option so that the user can alter window size. The control palette and the message buffer windows are moveable and can be placed as the user desires. A sample rearrangement is depicted in Figure 2.2.

To design a user interface, the application programmer chooses the edit design feature from the selection menu. A control can then be selected from the control palette window. The system asks the user to give the control a name and to specify any additional maximum, minimum, or initial values. The system then draws a floating picture of the control in the middle of the design window. The programmer

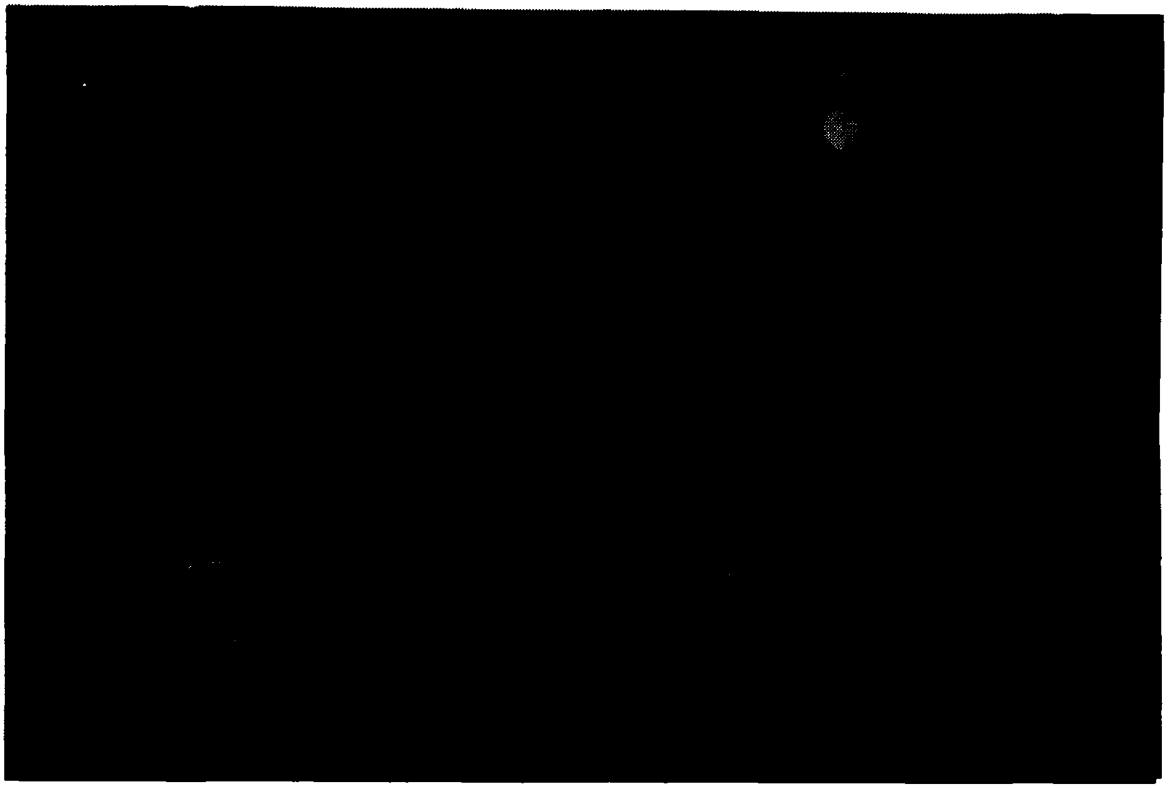

Figure 2.1 NPS Interface Builder Program Layout

uses the mouse to maneuver the floating control about the design window, allowing the determination of its placement. To drop the tool into its place in the window, the programmer clicks the middle mouse button, and a fixed picture of the control is drawn in the location specified. Figure 2.3 illustrates the design window with one fixed tool in place. By continuing to select tools from the tool palette window and dropping them into location in the design window, the application programmer can thus build a complete user interface. An interface created by this method is shown in Figure 2.4.

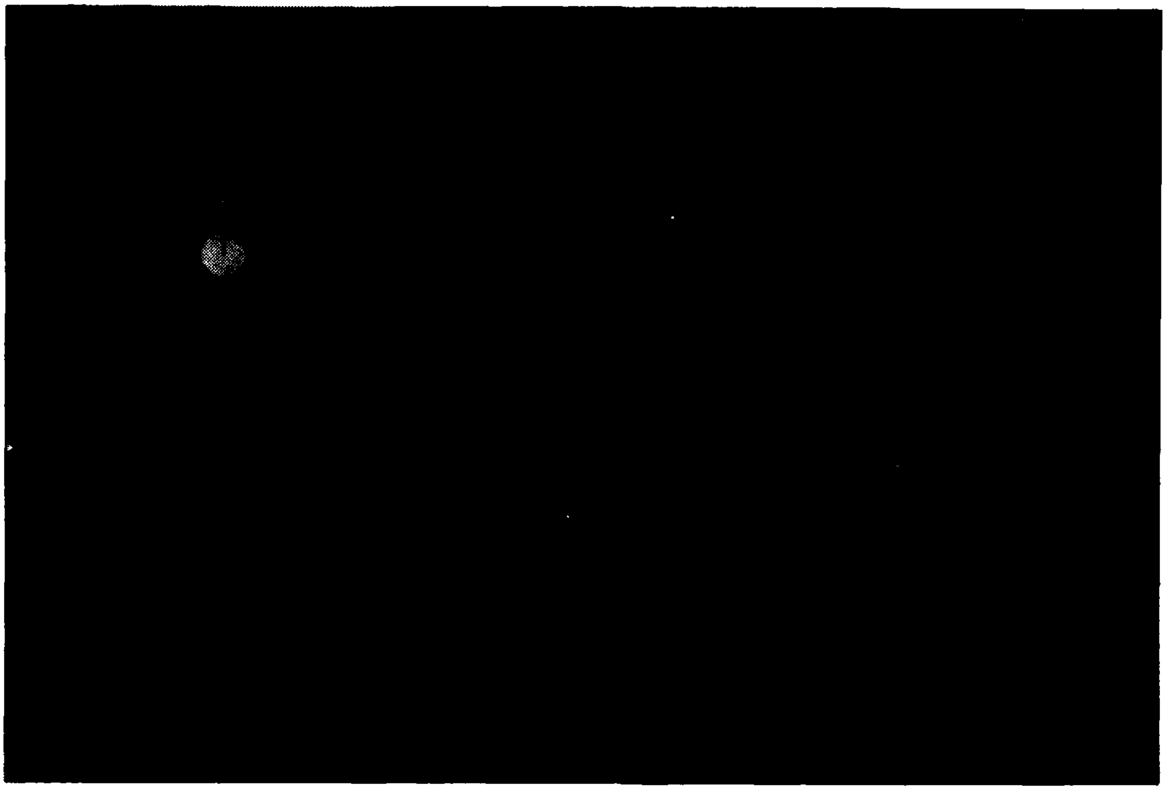

Figure 2.2 Sample NPS Interface Builder Window Rearrangement

Menu options include a move control selection, so that positions of fixed controls can be altered or adjusted if necessary, and a delete control selection, so that fixed controls can be eliminated from the design window entirely. The run controls option permits the programmer to test the operation of controls in the design window. The reset selection clears all fixed controls from the display window without saving any graphics information and is essentially akin to clearing a slate. The save design option writes the current state of the design window to two output files--an ASCII file listing of graphics information for each control,

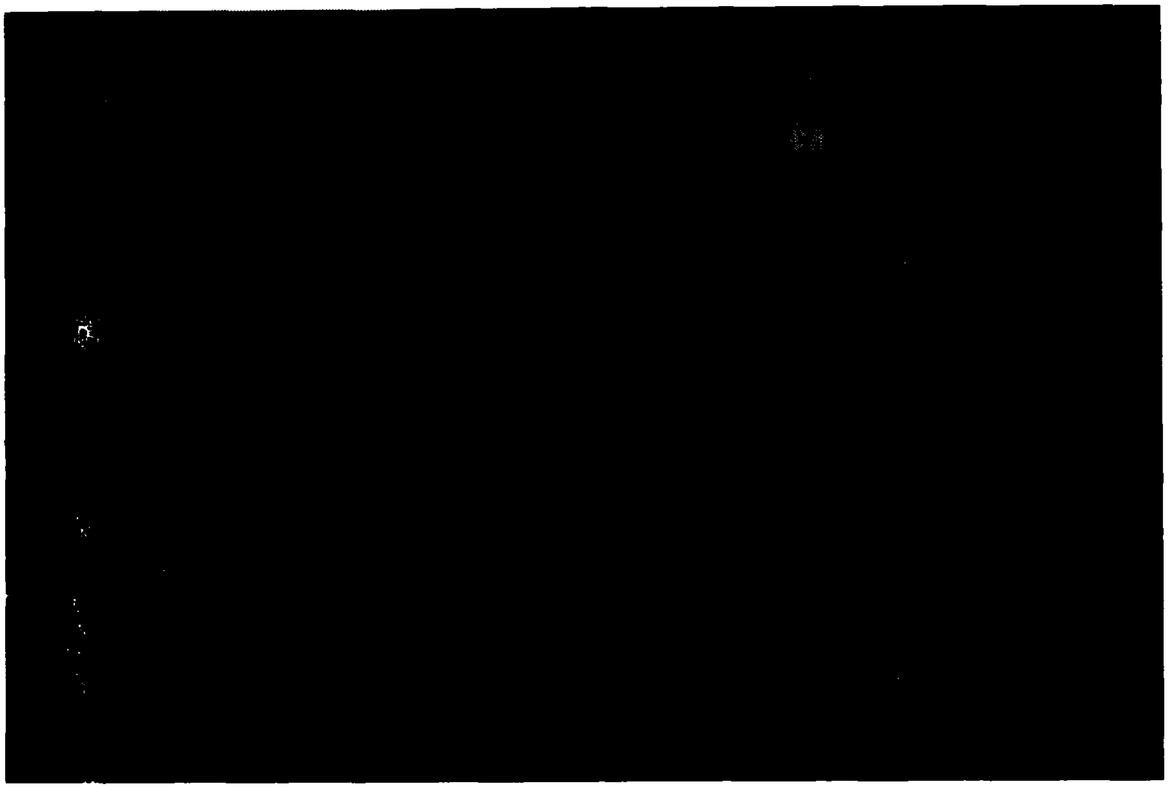

Figure 2.3 Design Window with Fixed Control

and a "C" language output file containing the skeleton of a graphics program that the user can compile, link, and execute. A sample ASCII file listing is presented in Figure 2.5, while a sample "C" language graphics program skeleton is illustrated in Figure 2.6. The ASCII file output provides the NPS IB program with its re-edit capability. The "C" language output file provides the user with a foundation for the application program.

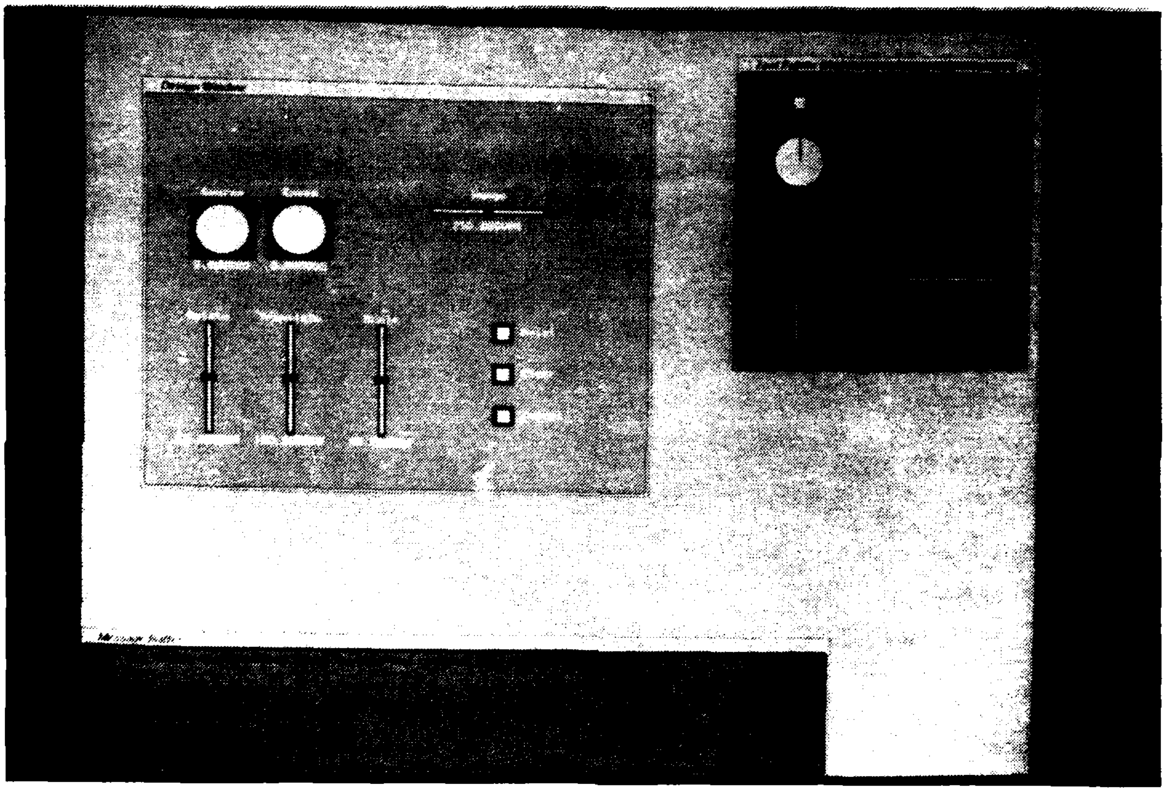

Figure 2.4 Sample Interface Created with NPS Interface Builder

```
/* This is the ASCII output file for a window created with
NPS Interface Builder */

/* save the window size */
lowerleftX 5
lowerleftY 5
upperrightX 1275
upperrightY 1004

/* toggle box information */
togglebox
title Cancel
controlid -1
controlnum 1
Xcoord 24.96
Ycoord 30.56
lowerleftxbound 23.78
lowerleftybound 29.31
upperrightxbound 26.15
upperrightybound 31.81
togboxend

/* dial information */
dial
title Speed
controlid -4
controlnum 2
Xcoord 13.90
Ycoord 49.20
maximumvalue 50.00
minimumvalue 0.00
lowerleftxbound 10.12
lowerleftybound 45.19
upperrightxbound 17.68
upperrightybound 53.21
dialend

/* vertical slider information */
verticalslider
title Rotate
controlid -5
controlnum 3
Xcoord 27.52
Ycoord 49.00
maximumvalue 360.00
minimumvalue 0.00
startvalue 180.00
lowerleftxbound 27.14
lowerleftybound 41.48
upperrightxbound 27.90
upperrightybound 56.51
vsliderend
```

Figure 2.5 NPS Interface Builder ASCII File Output Listing

```
#include"interfaceglobal.h"

main()
{
   short value;
   int wintoken;

   maketogbutton("Cancel",24.96,30.56,23.78,29.31,26.15,31.81,1);

   makedial("Speed",13.90,49.20,10.12,45.19,17.68,
                53.21,50.00,0.00,0.00,0.00,1);

   makevslider("Rotate",27.52,49.00,27.14,41.48,
                27.90,56.51,0.00,360.00,180.00,1);

   checkboxnum = 2;
   dialnum = 2;
   vslidernum = 2;

/* initialize the window */
wintoken = initwindow("custom window",5,5,1275,1004);

/* enter a drawing loop */
while(TRUE)
   {
   winset(wintoken);
   while(qtest())       /* loop while there is something on the queue */
      {
      switch(qread(&value))
         {
         case REDRAW:

                    winset(wintoken);
                    reshapeviewport();
                    winconstraints();
                    update_main_window();
                    break;

         case LEFTMOUSE:
                    if (value == 1)
                    {
                       runcontrols();
                       drawpicture();
                       swapbuffers();
                    }

                    break;
         case RIGHTMOUSE:

                    break;

         default:

                    break;

         }
      }
   drawpicture();
   swapbuffers();
   }
}
```

Figure 2.6 NPS Interface Builder "C" Language File Output Listing

III. THE NPS INTERFACE BUILDER PROGRAM

A. DESIGN ISSUES

The design of NPS IB encompassed two major concerns. The first area of concern involved the development of the user interface for the NPS IB program. The second area of concern involved the actual development and implementation of the program itself, including data structures and routines to support program functionality.

1. NPS Interface Builder User Interface

References 7 and 8 were consulted in an attempt to design the NPS IB program user interface in concert with recommended theories, principles, and guidelines of user interface design. The principle of consistency was the first and foremost principle considered, followed by the ability of the program to offer informative feedback and to offer simple error handling. In addition, two interaction styles were defined for the user interface, to include use of program menu selections and direct screen manipulation of objects of interest. The use of program menu selections structures user decision-making and supports program consistency, since each sequence of actions in the program must start with a menu selection. Each menu selection, then, is followed by the same set of actions in each selection category. In the edit design mode, for example,

the same sequence of actions is followed for each control selection: pick the control of interest, give the control name and value attributes, then place the control in its desired position in the design window.

The ability of the program to offer informative feedback and to offer simple error handling is made possible by both menu selection and direct manipulation of objects of interest. Since the user can point at control icons and can use the mouse to manipulate icons in the design window, the results of his actions are immediately observable. Simple error handling is provided by menu selections which allow the user to move or delete control icons from the design window, along with corresponding direct manipulation of icons to be moved or deleted. User feedback is also supported by the message buffer window, which provides instructions and verification information, such as confirmation that a control has been deleted.

A particular design issue of interest was the placement of the cursor and the floating control icon after its selection from the control palette window in the edit design mode option. Originally, the program design called for a click on the icon drawing of interest and for subsequent dragging of an icon clone from the control palette window into the design window. It was discovered, however, that this type of action between two concurrently running graphics windows was quite complicated, and the idea

was abandoned in favor of a new approach. The selected solution to the problem was to set the cursor to the middle of the design window and to use the mouse x and mouse y coordinates as the center x and center y for a drawing of the floating control. The user could then use the mouse to manipulate the cursor about the design window, with cursor movement mirrored by the floating control icon. The user is not as likely to be surprised by the unexpected cursor jump from the control palette window to the design window, since the action of selecting a control and placing it in the design window is broken up by the program request for user keyboard input. Since the user's hand must leave the mouse to operate the keyboard, the impact of the cursor jump from control palette window to design window is reduced considerably.

2. NPS Interface Builder Program Design

Three issues were considered in designing the structure of the NPS IB program. The first issue was that of saving graphics information for each control placed in the design window. This involved selecting an appropriate data structure for storing a variety of information about a single control. Data structures are discussed in the next part of this chapter. The second issue considered in designing the program was modularizing the program so that each group of routines in a single file performed one type of functionality for the program.

A final issue considered was the integration of program routines with the graphics program main event queue. It was decided that controls would be defined first, before entering the queue loop, in a window that would run concurrently with the design easel and the message buffer. Routines to process actions involving controls would be governed by global boolean variables inside the event queue loop. Display of controls drawn as a result of some user action would also take place within the event queue. Figure 3.1 illustrates basic program layout.

```
#define MAIN               /* define MAIN for global variable
                           recognition */
#include "interface.h" /* definitions for variables and
                           structures */

/* main driver routine */
main()
{
   ...(open main window)
   ...(open control palette window)
   ...(open message buffer window)
   ...(define controls)
   while (!exitprogram)
   ...(process event queue tokens)
   ...(redraw, if necessary)
   ...(change input window, if necessary)
   ...(process mouse hit, if necessary)
   ...(process routines to manipuiate controls)
   ...(send informative messages to message buffer window)
   ...(display controls)
   swapbuffers();
}
```

Figure 3.1 Program Layout Showing Included Files

B. DATA STRUCTURES

1. Saving Graphics Information

As noted in Section A of this chapter, a critical program design issue involved selection of a data structure that would hold graphics information for each control drawn in the design window. A record structure is chosen for this job, primarily because of its ability to group individual data elements of different data types as a single unit. This feature of the record structure is important since controls have a number of attributes of different data types, such as character type titles, floating point x center and y center coordinates, and integer identification numbers. The record structure is advantageous also because: (1) it allows the information about a control to be treated as a single data element, and (2) it permits separate use of any single data field within the record element itself. The ability to treat a data structure as a single element means that records can be manipulated in ways which correspond to the actual manipulation of control icons. Deleting a control icon, for instance, is as easy as eliminating the record structure for that particular icon. The ability to address single data fields of a record means that current values and coordinate values can be easily updated or changed.

The NPS IB program uses a different record structure for each control in the program. This is necessary since

different types of controls have different attributes. The record for a toggle box is displayed in Figure 3.2. The **title** field holds the name of the toggle box, as entered by the user. The **num** field holds a unique value which enables use of the IRIS picking mechanism since it becomes the loadname of the control. The **onoff** field represents a boolean value indicating whether the toggle box is turned on or off. The **Xcoord** and **Ycoord** fields hold the center x and center y coordinates which are determined when the tool is positioned in the design window. The **llxbd, urxbd, llybd,** and **urybd** fields represent the lower left x, upper right x, lower left y, and upper right y bounds of the control. These boundaries are used to identify controls in the design window when in the operate control mode.

```
/* define the structure which will hold graphics information
about toggle boxes */
typedef struct atogboxrecord {
    char title[50];   /* holds user-input title */
    int num;   /* unique number used as loadname of the
               control */
    int onoff;  /* indicates tog box on or off position */
    float Xcoord;   /* center x coordinate of the control */
    float Ycoord;   /* center y coordinate of the control */
    float llxbd;   /* lower left x boundary coordinate */
    float urxbd;   /* upper right x boundary coordinate */
    float llybd;   /* lower left y boundary coordinate */
    float urybd;   /* upper right y boundary coordinate */
    struct atogboxrecord  *next;   /* next pointer */
} togboxrecord, *ptogboxrecord;
```

Figure 3.2 Toggle Box Record Structure

A dial record is shown in Figure 3.3. Fields in the dial record with the same name as fields in the toggle box record also share the same functions. Four additional fields in the dial record require explanation. The **max** field is used to save the user-input maximum value for the control, while the **start** field is used to record the user-input minimum value. The **current** field holds the current value of the dial at any given time, and the **theta** field holds the current rotation angle of the dial needle.

```
/* define the structure which will hold graphics information
about dials */
typedef struct adialrecord {
    char title[50];    /* holds user-input title */
    int num;    /* unique number used as loadname of the
                control */
    float max;    /* holds user-input maximum value */
    float current;    /* holds control current value */
    float start;    /* holds user-input initial value */
    float theta;    /* holds current dial needle rotation
                    angle */
    float Xcoord;    /* center x coordinate of the control */
    float Ycoord;    /* center y coordinate of the control */
    float llxbd;    /* lower left x boundary coordinate */
    float urxbd;    /* upper right x boundary coordinate */
    float llybd;    /* lower left y boundary coordinate */
    float urybd;    /* upper right y boundary coordinate */
    struct adialrecord *next;    /* next pointer */
} dialrecord, *pdialrecord;
```

Figure 3.3 Dial Record Structure

The record for a vertical slider is illustrated in Figure 3.4. Again, fields in the vertical slider record with the same names as fields in the dial and toggle box records also share the same functions. Two additional

fields in the vertical slider record require explanation. The **min** field holds the user-input minimum value for the slider, and the **position** field holds the center x coordinate of the slider button.

```
/* define the structure which will hold graphics information
about vertical sliders */
typedef struct averticalsliderecord {
    char title[50];   /* holds user-input title */
    int num;   /* unique number used as loadname of the
               control */
    float max;   /* holds user-input maximum value */
    float min;   /* holds user-input minimum value */
    float current;  /* holds current value of the control */

    float start;   /* holds user-input initial value */
    float Xcoord;   /* center x coordinate of the control */
    float Ycoord;   /* center y coordinate of the control */
    float position;   /* holds current position of slider
                      button */
    float llxbd;   /* lower left x boundary coordinate */
    float urxbd;   /* upper right x boundary coordinate */
    float llybd;   /* lower left y boundary coordinate */
    float urybd;   /* upper right y boundary coordinate */
    struct averticalsliderecord *next;   /* next pointer */
} verticalsliderecord, *pverticalsliderecord;
```

Figure 3.4 Vertical Slider Record Structure

The horizontal slider record structure is an exact duplicate of the vertical slider record and is not reprinted here. The horizontal and vertical slider record structures could have been combined into one set of records, however, it was decided that the program made more logical sense if each control had its own grouping of records.

2. The Linked List Structure

Each control record structure contains an additional field called **next,** which holds a pointer to another record structure of similar type. Linking the records together in this way is advantageous for several reasons. Control records can easily be added to or deleted from the linked list. The linked list structure eliminates the need for keeping global counters that would dictate how many times to call the drawing routine for a particular control. The count is provided instead by a traversal of the linked list, which visits each control record exactly once. The linked list structure provides control over the composition of the design window, since pointers can be rerouted to eliminate drawing of selected controls. This scheme is used to move controls in the design window. Pointers are temporarily routed around the control to be moved while its new placement is determined. When the control is fixed in place, the pointers are reconnected and the control is drawn in turn when the list is traversed.

C. SYSTEM STARTUP

The executable program file is **interface.** While in the directory containing the executable file, type **interface** to start the program. Three windows open and run concurrently during NPS IB operation.

The design window opens to full screen size, but can be immediately resized if the user desires. The control

palette window opens in the upper right hand corner of the screen and draws its inventory of controls. The message buffer window opens in the lower left hand corner of the screen and displays opening messages and advice. The locations of both the control palette and message buffer windows can be moved to different positions if the user so desires.

The user must select an option from the system menu in order to operate the program. The system menu is shown in Figure 3.5. The system menu is a popup menu with roll-off-the-side options, and is invoked by depressing the right mouse button. Releasing the right mouse button on the menu selection desired prompts the system to execute the action for that selection. The user can select any menu option at any time during program execution. If the user chooses a menu option that is not meaningful, however, no action is taken. The save design option chosen at this point writes only the lower left x and y coordinates and the upper right x and y coordinates of the design window to its output files.

D. MAIN DRIVER ROUTINE

The NPS IB main driver routine is located in the file **interface.c.** The main driver routine directs program action based on user menu selection. Evaluation of user menu selection is made in the routine **handletoken,** located in the file **handletoken.c.** The routine **handletoken** switches on the

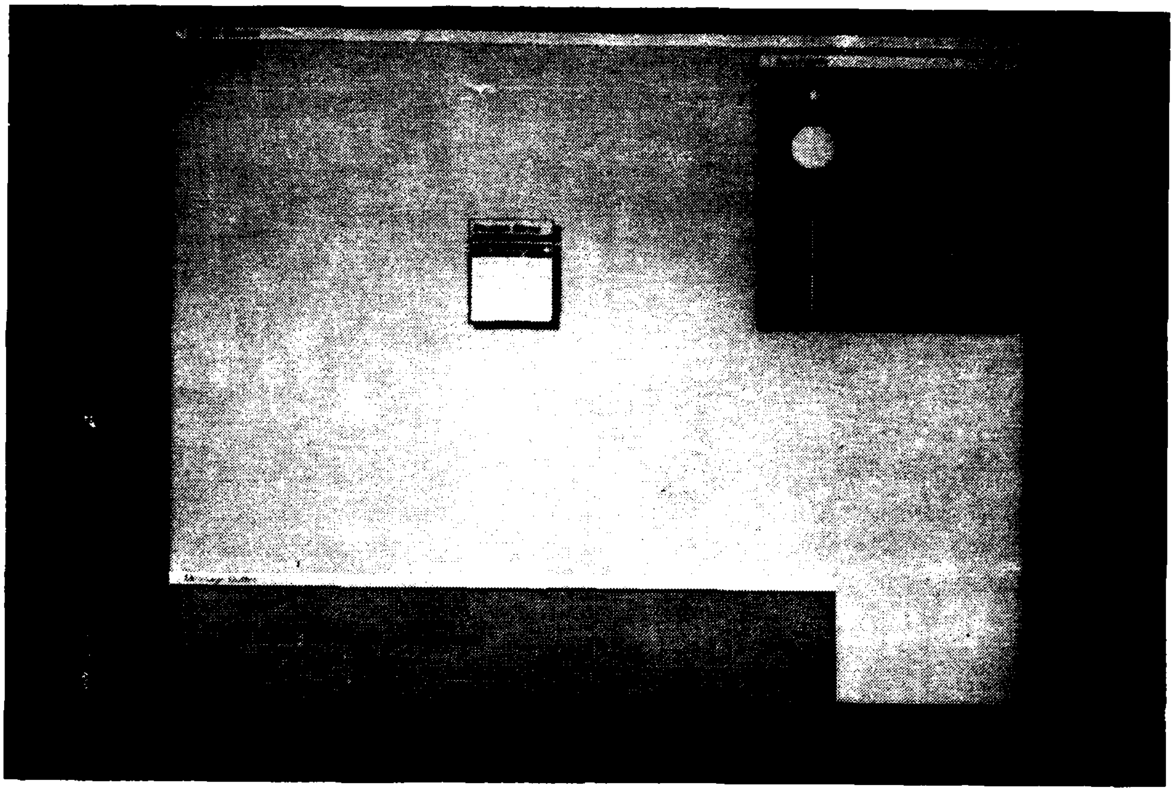

Figure 3.5 NPS Interface Builder System Menu

value of the menu selection, sets appropriate global boolean variables based on the user menu selection, sends appropriate messages to the message buffer window, and returns to the main routine. As the main routine cycles through the event loop, if-loop constructs governed by the global boolean variables are evaluated. On evaluation of a true boolean variable, the if-loop is entered and routines performing the action corresponding to the user menu selection are executed. A switch statement might have been used here with equal effectiveness.

Each cycle through the event loop also draws the three concurrently running windows. This is to ensure that the keyboard input window, which pops up when user input is required and closes when user input is complete, is fully erased from any of the window which it might overlay.

The majority of constant and global variable definitions are located in the include file **interface.h.** Control record structures are also defined in this file. Global variables defined in **interface.h** include those which govern program action and which establish size for control icon drawing. A small number of global constant definitions are located in the include file **sbglobal.h.** The original version of this file belongs to the S & B package, which was described in Chapter I. The **sbglobal.h** file became part of the NPS IB program because original plans for the program called for integration of the S & B package as a feature that would quickly create button and slider windows, however, this feature of the NPS IB program was never fully implemented. Although this feature of the program was never realized, the **sbglobal.h** file, with a few modifications, remains a part of the NPS IB. The **sbglobal.h** file contains constant definitions for color and size of program icon drawings.

E. REDRAW/RESIZE CODE

When the NPS IB program executes, the design window opens to full screen size. In the original program design,

the full screen design window was the only size design window offered. This was considered too constraining for a user, however, who might have need for a variety of sizes of control windows. It was decided to offer the user an option to resize the design window, and to enable the user to move the control palette and message buffer windows to new positions as well.

The user can select the resize or move option by using the right mouse button to pull down the menu from the title bar of the window in question. This pull down menu is not a product of the NPS IB program, but rather is a creation of the window manager. The selection of a resize or move menu option generates a program REDRAW token which must then be processed by the main event queue. The NPS IB program contains code which will relocate and redraw windows when the REDRAW token is received. This code is drawn from the S & B package program.

When the resize option is indicated for the design window, the user manipulates the mouse to interactively sweep out a new size for the window. The click of any mouse button drops the newly sized window into place. Resize code for the design window records the new window coordinates and uses them to compute new x and y ratios of world to screen coordinates. The new ratio values in turn are used to recalculate global variables for drawing controls. This code is illustrated in Figure 3.6. Recalculating the global

```
/* this routine calculates global coordinates for tool drawing */
setglobalcoordinates(lldx,lldy,urdx,urdy)
int lldx,lldy,urdx,urdy;
{
   set_scaleX = 120.0/((float)(urdx)-(float)(lldx));
   set_scaleY = 100.0/((float)(urdy)-(float)(lldy));

   /* determine width of tog button */
   togbuttonwidth = BUTTONWIDTH * set_scaleX;
   togbuttonwidth2 = togbuttonwidth/2;

   /* determine height of tog button */
   togbuttonHT = BUTTONHT * set_scaleY;
   togbuttonHT2 = togbuttonHT / 2;

   /* determine outline parameters of tog button */
   togbutton_outlineX = BUTTONOUTLINEWIDTH * set_scaleX;
   togbutton_outlineY = BUTTONOUTLINEWIDTH * set_scaleY;

   /* determine width of dial box */
   dialwidth = DIALWIDTH * set_scaleX;
   dialwidth2 = dialwidth/2;

   /* determine height of dial box */
   dialHT = DIALHT * set_scaleY;
   dialHT2 = dialHT/2;

   /* determine dial outline parameters */
   dialoutlineX = DIALOUTLINEWIDTH * set_scaleX;
   dialoutlineY = DIALOUTLINEWIDTH * set_scaleY;

   /* determine dial radius */
   insideradius = dialwidth2 - dialoutlineX - 0.3;
   outradius = dialwidth2 - dialoutlineX - 0.1;

   /* determine width of vertical slider bar */
   vertbarwidth = BARWIDTH * set_scaleX;
   vertbarwidth2 = vertbarwidth / 2;

   /* determine height of vertical slider bar */
   vertbarHT = BARHT * set_scaleY;
   vertbarHT2 = vertbarHT / 2;

   /* determine width of slider button */
   vertsliderwidth = SLIDERWIDTH * set_scaleX;
   vertsliderwidth2 = vertsliderwidth/2;

   /* determine height of slider button */
   vertsliderHT = SLIDERWIDTH * set_scaleY/2;
   vertsliderHT2 = vertsliderHT/2;

   /* determine width of slider bar inside */
   vertinsidebarwidth = InsideBARWIDTH * set_scaleX;
   vertinsidebarwidth2 = vertinsidebarwidth/2;

   /* determine width of horizontal slider bar */
   hzbarwidth = BARHT * set_scaleX;
   hzbarwidth2 = hzbarwidth/2;

   /* determine height of horizontal slider bar */
   hzbarHT = BARWIDTH * set_scaleY;
   hzbarHT2 = hzbarHT/2;

   /* determine width of slider button */
   hzsliderwidth = SLIDERWIDTH * set_scaleX/2;
   hzsliderwidth2 = hzsliderwidth/2;

   /* determine height of slider button */
   hzsliderHT = SLIDERWIDTH * set_scaleY;
   hzsliderHT2 = hzsliderHT/2;

   /* determine width of slider bar inside */
   hzinsidebarwidth = InsideBARWIDTH * set_scaleY;
   hzinsidebarwidth2 = hzinsidebarwidth/2;
}
```

Figure 3.6 Code to Set Global Coordinates for Drawing Control Icons

variables based on new x and y ratios is necessary to prevent distortion of icon drawings in the resized design window.

Drawing of controls in the main window is accomplished in the world coordinate system. This was done to facilitate integration of NPS IB with the S & B package program, which draws sliders and buttons in world coordinate system coordinates. Although the S & B package program was never fully integrated with NPS IB, the world coordinate system remained the basis for NPS IB control drawings. Unfortunately, using the world coordinate system means that some distortion of the dial icon may occur if the design window swept out by the user does not approximate a square. This occurs because world coordinates may be stretched longer along one axis of the design window if the window does not approximate a square. This has the effect of stretching the dial circle in the direction of the long axis, making it look more oval than circular. This problem can be resolved by adjusting the world coordinate system to compensate for the longer window axis.

Resize and redraw code works the same way for the control palette window as for the design window. New window coordinates are obtained, ratios calculated, global variables computed, and the window is redrawn. Redraw code for the message buffer is much simpler, however, since this window always remains the same size. To effect a move of

the message buffer window, it is necessary to obtain the lower left x and lower left y coordinates of the new window position and enter them into the window record structure. The message buffer window is then drawn in its new position.

F. DRAWING ALGORITHMS

1. Control Palette Window Drawings

Four controls are drawn in the control palette window after the main routine opens the window up. These four controls are a toggle box, a dial, and two versions of a slider, one oriented lengthwise along the y axis and the other oriented lengthwise along the x axis.

Code for drawing the control palette window is located in the file **tool_window.c**. First, global variables for drawing control icons are calculated based on world to screen ratios as described in the previous section of this chapter. Then the center x and center y of each control are established. The control center x and center y are fixed points specified in the global include file **tglobal.h**. This means that positions of controls in the window can be altered by editing the **tglobal.h** file rather than by accessing the drawing code directly. A single pass through the **drawtools** routine then draws each control icon. For each rectangular portion of each control drawn, the lower left corner and upper right corner points are calculated, utilizing center points and global size variables. Text drawing is accomplished in the same way. For circle

drawing, the center x and y points are given in **tglobal.h** and the radius is calculated based on the size of the dial outline box. A control identification number is loaded onto the names stack as the largest portion of each control is drawn. This number identifies the type of control for the IRIS picking mechanism.

2. Design Window Drawing

Code for drawing control icons in the design window is located in the file **draw.c.** The code in the file handles two cases of drawing controls in the design window. The first case involves the drawing of a floating control tied to the mouse cursor, which the user can then move about the design window to determine control placement. The second case involves drawing a fixed control which appears in the design window after the user clicks the middle mouse button to set placement of the floating control.

Four separate routines accomplish the drawing of the four controls. The separation of control drawing routines permits the use of the same drawing code for drawing of both floating and fixed controls. Figure 3.7 shows the code used to select the appropriate drawing routine for the floating control icon. The routine switches on the variable **tool-picked,** which holds the identification number of the type of control selected from the control palette window. The case statement is entered when **toolpicked** matches a control identification number, and subsequently the appropriate

```
/* this routine determines which floating tool is to be
drawn */
drawthefloatingtool(fx,fy)
float fx,fy;
{
  switch(toolpicked)
  {
    case TOGBOX:

      /* draw a toggle box */
      drawfloat1 = TRUE;
      drawthetogbox(fx,fy,titlebuffer);

      break;

    case DIAL:

      /* draw a dial */
      drawfloat2 = TRUE;
      drawdial(fx,fy,titlebuffer,curnum);

      break;

    case VSLIDER:

      /* draw a vertical slider */
      drawfloat3 = TRUE;
      drawvertslider(fx,fy,slidertitle,curnum);

      break;

    case HSLIDER:

      /* draw a horizontal slider */
      drawfloat4 = TRUE;
      drawhorizslider(fx,fy,slidertitle,curnum);

      break;

    default:

      qdevice(INPUTCHANGE);
      qdevice(RIGHTMOUSE);
      qdevice(LEFTMOUSE);

      break;
  }
}
```

Figure 3.7 Program Code for Drawing Floating Control

control drawing routine is called. Because the same code is used for drawing floating and fixed controls, it became necessary to number the **drawfloat** variable in order to prevent side effects to the other controls when any floating control was being drawn. By numbering the **drawfloat** variable, it was possible to draw just a floating dial, for example, and still cycle through the drawing code for the other controls without any adverse affects, even though that code also contained provisions for drawing floating controls. Figure 3.8 illustrates the code used to draw fixed controls in place in the design window. After a control is selected from the control palette and dropped into place in the design window, a record structure is created for the control and entered into the control linked list. This reduces the drawing algorithm to a traversal of four linked lists, with the proper drawing routine called for each control encountered in the traversal. As each control in the list is traversed, pointers access the information for that control and provide the information to the drawing routine.

Each control drawing routine is configured to handle the drawing of a floating or fixed control. The routine for drawing a toggle box is shown in Figure 3.9 as an example. If a floating control is to be drawn, the routine is entered with the x and y coordinates of the cursor, which at this point has been placed in the center of the design window.

```
/* this routine draws the design window picture by calling
individual tool
drawing routines */
drawthepicture()
{
  /* for the number of toggle boxes we have ... */
  for (togboxcurrent = togboxhead; togboxcurrent != NULL;
            togboxcurrent = togboxcurrent->next)

  /* ... draw them */
  drawthetogbox(togboxcurrent->Xcoord,togboxcurrent->Ycoord,
            togboxcurrent->title);

  /* for the number of dials we have ... */
  for (dialcurrent = dialhead; dialcurrent != NULL;
dialcurrent = dialcurrent->next)

  /* ... draw them */
  drawdial(dialcurrent->Xcoord,dialcurrent->Ycoord,dialcur-
rent->title, dialcurrent->current);

  /* for the number of vertical sliders we have ... */
  for (verticalslidercurrent = verticalsliderhead;
verticalslidercurrent != NULL; verticalslidercurrent =
 verticalslidercurrent->next)

  /* ... draw them */
  drawvertslider(verticalslidercurrent->Xcoord,
verticalslidercurrent->Ycoord,verticalslidercurrent->title,
            verticalslidercurrent-> current);

  /* for the number of horizontal sliders we have ... */
  for (horizslidercurrent = horizsliderhead;
horizslidercurrent != NULL; horizslidercurrent =
            horizslidercurrent ->next)

  /* ... draw them */
  drawhorizslider(horizslidercurrent->Xcoord,
horizslidercurrent-> Ycoord,horizslidercurrent-> title,
            horizslidercurrent-> current);

}   /* end drawthepicture */
```

Figure 3.8 Program Code for Drawing Fixed Controls

```
/* this routine draws the checkboxes in the main window */
drawthetogbox(tx,ty,string)
char *string;
float tx,ty;
{
   float x1,x2,y1,y2;

   /* save the matrix on top of the stack */
   pushmatrix();

   if (drawfloat1)
   {
      /* translate the tool to where we want it */
      translate(tx,ty,0.0);

      /* need to draw the tool at 0.0 so it may be properly translated */
      tx = 0.0;
      ty = 0.0;
   }

   /* draw the button outline */
   x1 = tx - togbuttonwidth2;
   x2 = x1 + togbuttonwidth;
   y1 = ty - togbuttonHT2;
   y2 = y1 + togbuttonHT;
   RGBcolor(BUTTONOUTLINE);
   if (!drawfloat)
      loadname(togboxcurrent->num);
   rectf(x1,y1,x2,y2);

   /* draw the button inside */
   x1 = x1 + togbutton_outlineX;
   x2 = x2 - togbutton_outlineX;
   y1 = y1 + togbutton_outlineY;
   y2 = y2 - togbutton_outlineY;
   if (drawfloat1)
   {
      RGBcolor(BUTTONINSIDEOFF);
      rectf(x1,y1,x2,y2);
   }
   else
   {
      if (togboxcurrent->onoff == ON)
      {
         RGBcolor(BUTTONINSIDEON);
         rectf(x1,y1,x2,y2);
      }
      else
      {
         RGBcolor(BUTTONINSIDEOFF);
         rectf(x1,y1,x2,y2);
      }
   }

   /* draw the button title */
   RGBcolor(TITLECOLOR);
   x1 = tx + 25 * set_scaleX;
   y1 = ty - togbuttonHT2 * set_scaleY;
   cmov2(x1,y1);
   charstr(string);

   /* restore the top of the matrix stack */
   popmatrix();

   drawfloat1 = FALSE;
}
```

Figure 3.9 Toggle Box Drawing Routine

The drawing matrix is saved on top of the stack, and the routine if-loop construct is entered. In the if-loop construct, the computer is instructed to translate the drawing of the control to the position of the cursor. This ties the drawing of the control to the position of the cursor and creates the floating effect of the control. The x and y coordinates passed into the routine are then set to zero, and become the center x and center y for drawing of the control. This means that the icon is drawn at (0.0,0.0,0.0) and then translated to the position of the cursor. The top of the matrix stack is then popped.

If a fixed control is to be drawn, the drawing routine is entered with the control record x and y coordinates, which were saved when the user dropped the floating control into the design window. The if-loop construct of the routine is skipped, eliminating the translation portion of the routine. The saved x and y coordinates are then used to calculate coordinates for control drawing. In addition, a unique identification number is loaded with the largest portion of the fixed control drawn. This number is used to identify the control for the IRIS pick mechanism.

G. EDIT DESIGN MODE

1. <u>IRIS Picking Mode</u>

The IRIS picking mechanism can be used to identify objects on the screen that appear near the cursor. When the

picking mechanism is invoked, the system loads the product of the window projection transformation matrix and the picking matrix at the top of the matrix stack. This maps objects to be picked to their proper coordinates. The system then records hits for any routines that draw into the picking region [Ref. 9].

2. Control Selection

The IRIS picking mechanism is used to select controls in the control palette window. When a control is picked, its identification number is loaded into the variable **toolpicked.** The user is then asked to give the control name and value attributes. The code to perform this action is located in the file **process.c.** The **nametool-picked** routine switches on the value of **toolpicked** in order to make proper queries to the user for obtaining control information. The **nametoolpicked** routine calls the keyboard input routine in order to permit users to enter information. When proper information for the control has been obtained, the system draws a floating control as described in Section F of this chapter.

3. Creating Record Structures for Saving Graphics Information

When the user clicks the middle mouse button to drop a control into the design window, a record structure is allocated for the control, and all graphics information is immediately recorded into the proper field. The code for accomplishing this is also located in the file **process.c.**

Figure 3.10 shows creation of a record structure and saving of graphics information for a toggle box. The code illustrated also shows the insertion of the record into the linked list. If it is the first record in the list, the head and tail pointers are set to that record and the next pointer is set to NULL. If the record is not first in the list, the record is just appended to the list and the tail pointer is adjusted to that record. This process works the same way for the other controls, thus code for those operations is not reprinted here.

H. MOVE CONTROL MECHANISM

The NPS IB program would be terribly constraining if users were locked in to any one placement for a control. The move control mechanism gives a user the ability to move controls that have already been placed in the design window. The routines that provide NPS IB with its move capability are located in the file **move.c.** In addition, **move.c** contains the routine which will delete a control from the design window.

The IRIS picking mechanism is used to select the control to be moved in the design window. The identification number of the control selected is read into the variable **tool-picked.** The routine **getcontrolid** searches through record structures in each linked list, attempting to match the number in the variable **toolpicked** with the identification number saved in each record structure. Figure 3.11 shows a

```
/* allocate memory for this guy */
cbr = (togboxrecord*)malloc(sizeof(togboxrecord));

/* store the graphics information */
strcpy(cbr->title,titlebuffer);
cbr->num = controlindex;
cbr->Xcoord = sx;
cbr->Ycoord = sy;
cbr->llxbd = sx - togbuttonwidth2;
cbr->urxbd = (sx - togbuttonwidth2) + togbuttonwidth;
cbr->llybd = sy - togbuttonHT2;
cbr->urybd = (sy - togbuttonHT2) + togbuttonHT;
cbr->onoff = OFF;
cbr->next = NULL;

/* put the guy in the list */
if (togboxhead == NULL)
{
  /* this is the first guy in the list */
  togboxhead = togboxtail = cbr;

  /* set tail pointer to null */
  togboxtail->next = NULL;
}
else
{
  /* set the pointer for the last guy in the list to the new
guy */
  togboxtail->next = cbr;

  /* now reset the tail pointer */
  togboxtail = cbr;
}
```

Figure 3.10 Record Creation and List Maintenance for a Toggle Box

code segment from the routine used to accomplish this. If the control is found, the identification number in the variable **toolpicked** is replaced by a second number identifying the type of control found. This is because the variable **toolpicked** will be used to determine the type of floating control icon to be drawn in the design window when

```
/* this routine identifies the tool found in the pick */
getcontrolid()
{
  int found = FALSE;

  /* set previous and current list pointers to head */
  togboxprevious = togboxcurrent = togboxhead;
  verticalsliderprevious = verticalslidercurrent =
verticalsliderhead;
  horizsliderprevious = horizslidercurrent =
horizsliderhead;
  dialprevious = dialcurrent = dialhead;

  /* loop through all the lists until we find the guy ... */
  while ((!found) && (toolpicked > 0))
  {
    /* search the list of toggle boxes */
    while ((togboxcurrent != NULL) && (!found))
    {
      if (togboxcurrent->num == toolpicked)
      {
        toolpicked = TOGBOX;
        strcpy(titlebuffer,togboxcurrent->title);
        if (togboxcurrent == togboxhead)
          togboxhead = togboxhead->next;
        togboxtemp = togboxcurrent;
        found = TRUE;
      }
      else
      {
        togboxprevious = togboxcurrent;
        togboxcurrent = togboxcurrent->next;
      }
    }
    ...(search remaining list of controls)
  } /* end while */
} /* end getcontrolid */
```

Figure 3.11 Searching Linked Lists for a Control

the move is to occur. Any graphics information necessary for drawing the floating control is transferred from the record to buffer variables.

If the control to be moved has a record structure at the head of a linked list, the head pointer is advanced to the

next record in the list. This has the effect of removing the icon drawing from the design window, since the control drawing algorithm is regulated by a traversal of the linked list, starting from the head pointer. If the control is not at the head of a list, the pointer from the structure preceding the control to be moved is simply redirected to point at the structure following the control to be moved. This has the effect of erasing the control from the design window since the node of the control to be moved is not visited when the linked list is traversed for drawing.

Erasure of the control to be moved corresponds with the appearance of a floating control in the design window. The floating control is labeled with all the attributes of the previously fixed control. The user manipulates the floating control to a new position, then presses the middle mouse button to drop the moved control into place. Drawing the moved control is simply a matter of saving the new graphics information and then replacing or reconnecting pointers, as illustrated in the code segment of Figure 3.12. If the head pointer was moved, it is replaced at the head of the list. Otherwise, the pointer preceding the structure for the moved control is reconnected to the moved control structure. The delete control mechanism is an extension of the move control mechanism. Pointers are redirected in the same manner as for the move control mechanism. But instead of reconnecting the record structure when the middle mouse is clicked, the

```
/* get the new control coordinates */
togboxtemp->Xcoord = sx;
togboxtemp->Ycoord = sy;
togboxtemp->llxbd = sx - togbuttonwidth2;
togboxtemp->urxbd = (sx - togbuttonwidth2) + togbuttonwidth;
togboxtemp->llybd = sy - togbuttonHT2;
togboxtemp->urybd = (sy - togbuttonHT2) + togbuttonHT;

/* if the control moved had the head pointer, restore it */
if (togboxprevious->next == togboxhead)
  togboxhead = togboxprevious;
else
  togboxprevious->next = togboxtemp;
```

Figure 3.12 Code for Reconnecting a Moved Control Structure in the Linked List

pointers remain in the new configuration. The tool is erased from the design window and is not replaced. The memory occupied by the old record structure is freed and made available for other use.

I. RUN CONTROL MECHANISM

The run control mechanism provides a user with the ability to check the operation of controls in the design window. To invoke the run control mode, the user selects the run controls option from the program main menu. The user can then use the left mouse button to operate any control in the design window. The run control option is deselected when another menu option is chosen. The code for the run control option is located in the file **run.c.**

When the run control mode is invoked, the mouse x and mouse y coordinates are read. The **operatecontrol** routine then traverses the four linked lists and compares the mouse

x and mouse y locations with the boundary limits for each control. If the cursor lies within the boundary limits of a control, that particular control is operated. Figure 3.13 is a code segment illustrating operation of a toggle box.

```
/* get the world coordinates from the mouse */
wx = ((float)(getvaluator(MOUSEX))-dllx)*set_scaleX;
wy = ((float)(getvaluator(MOUSEY))-dlly)*set_scaleY;

/* now traverse the linked lists to see if we are in within
 the bounding box of a control */

/* for the number of toggle boxes we have . . . */
for (togboxcurrent = togboxhead; togboxcurrent != NULL;
          togboxcurrent = togboxcurrent->next)
{
  /* do the boundary test */
  if ((wx > togboxcurrent->llxbd) && (wx < togboxcurrent->
urxbd) &&
      (wy > togboxcurrent->llybd) && (wy < togboxcurrent->
urybd))
  {
    /* switch the control to the other position, based on
   what position its already in */
    if (togboxcurrent->onoff == OFF)
      togboxcurrent->onoff = ON;
    else
      togboxcurrent->onoff = OFF;
  }
}
```

Figure 3.13 List Traversal and Toggle Box Operation

IV. NPS INTERFACE BUILDER MESSAGE BUFFER

A. PURPOSE OF THE MESSAGE BUFFER

The NPS IB message buffer window gives the program its help capability. The message buffer window provides help by displaying instructions designed to guide a user through accomplishment of a task made by menu selection. The message window also provides verification information by sending a message to the screen when some action has been completed, such as moving or deleting a control. The message buffer window gives the NPS IB a means of providing more informative feedback to the user.

B. OPERATION OF MESSAGE BUFFER WINDOW

Definitions of structures that support operation of the message buffer are located in the file **mess.h**. The structure which defines a line of text to be displayed in the buffer is illustrated in Figure 4.1. The structure consists of two pointers; the pointer called **ptr** is directed to a string of text which constitutes a line, and the pointer called **next** is directed to the next line in the buffer. The structure which holds attributes of the message buffer window is shown in Figure 4.2. The variables **nlines** and **nchars** hold the number of lines in the buffer and the maximum line length for the buffer respectively. These values will be used to compute height and width for setting

```
/* define a type and structure for each line of the ring
buffer that is the message buffer display.
*/
struct lineinbuf
{

   char *ptr;    /* ptr to the string for this line */

   struct lineinbuf *next;  /* ptr to next line in the
                               buffer */

};

typedef struct lineinbuf LINEINBUF;    /* define a line in
                                          the buffer type */
```

Figure 4.1 Structure that Defines a Line of Text

the size of the message buffer window. The pointers **topline** and **lastline** point to lines of display buffer text, with topline directed at the first line of the display and lastline directed at the last line of the display. The field **winname** holds the name of the window opened for the message buffer, and the fields **x** and **y** hold the lower left corner value of the window. **Textcolor** defines the color of the message buffer text, while **backcolor** holds the color of the background of the message buffer window. Finally, the fields **width** and **height** hold the computed values of width and height for the window.

The routines which create and operate the message buffer window are located in the file **mess.c**. These routines open the message buffer window and govern the operation of the ring buffer. The ring buffer provides the mechanism that gives the message buffer window its appearance of scrolling

```
/* define a type and structure for the message buffer */
struct messbuf
{

   long nlines;   /* number of lines in the buffer */

   long nchars;   /* maximum line length for the buffer */

   LINEINBUF *topline;   /* ptr to top line of the display
                          */

   LINEINBUF *lastline; /* ptr to last line of the display
                         */

   long winname;     /* the name of the window opened for
                        this message buffer.
                     */

   long x,y;         /* lowerleft corner of ther message
                     buffer */

   short textcolor; /* color of the text in the message
                    buffer */

   short backcolor; /* color of the background in the
                    message buffer */

   long width;       /* width in pixels of the message buffer
                     */

   long height;      /* height in pixels of the message
                     buffer */

};

typedef struct messbuf MESSBUF;   /* define type MESSBUF
(need one of these for every defined message buffer) */
```

Figure 4.2 Structure that Defines Message Buffer Window

text. In the routine **createmessbuf,** memory is dynamically allocated to accommodate the structure and values are read in for the size, location, and color of the window. Then the ring buffer is created. The code in Figure 4.3 shows how the ring buffer is set up. For the number of lines

```
/* we are going to build a circularly linked list with links
in only 1 direction.
*/

/* say there is no previous guy */
prev = (LINEINBUF *)NULL;

/* build the list backwards */
for(i=0; i < ilines; i=i+1)
{

  temp = prev; /* hold the previous guy's loc */

  /* get space for this line.
     if i == 0, this guy is the last link of the list.
  */
  prev = (LINEINBUF *)malloc(sizeof(LINEINBUF));

  if(i == 0)
  {

    /* save prev as lastline of the buffer */
    mess->lastline = prev;

  }

  /* set the next field at held previous guy */
  prev->next = temp;

  /* get space for the line of text plus 1 */
  prev->ptr = (char *)malloc((ichars+1));

  /* stick chars into the string */
  strcpy(prev->ptr,"");

}   /* endfor all lines in the buffer */

/* at the end of the for loop, prev == 1st line. */
mess->topline = prev;

/* must make last line point back at first line */
mess->lastline->next = mess->topline;
```

Figure 4.3 Code for Constructing Ring Buffer

specified, memory is allocated for the LINEINBUF structures. The first structure created gets the lastline pointer. As

the routine cycles through the loop, it also allocates space for text and initializes text fields to empty characters. The **topline** pointer is set to the last structure created, and then the **lastline next** pointer is directed at the **topline** node, thus completing the ring. The window and ring buffer are then ready for display.

The display routine sets the message buffer window as the current graphics window, then clears the background and draws the window border. The routine then cycles through the ring buffer list and subsequently displays each line of text as saved in each LINEINBUF structure. The display routine is called once, near the end of the main driver program. This means that lines added to the message buffer routine outside the main driver are not displayed until the return to main.

Adding a line of text to the ring buffer for display in the message buffer window is mostly a matter of manipulating the **topline** and **lastline** pointers. This code is shown in Figure 4.4. The **topline** pointer is advanced to the next node in the list. The **lastline** pointer is also advanced to the next node in the list, which is the old position of **topline.** New text is then copied into the **ptr** field of the node pointed to by **lastline.** This has the effect of wiping out the old text, which was the previous first line, and replacing it with new text in a line at the bottom.

```
addlinetomessbuf(mess,str)

MESSBUF *mess;    /* ptr to message buffer */

char str[];       /* character string */

{

   /* push topline ahead to next guy */
   mess->topline = mess->topline->next;

   /* push lastline ahead to next guy */
   mess->lastline = mess->lastline->next;

   /* stick in the line at lastline */
   if(strlen(str) <= mess->nchars)
   {
      strcpy(mess->lastline->ptr,str); /* just stick in the
                                          string */
   }
   else
   {
      /* only copy the first bytes up to the end of the line
available */
      strncpy(mess->lastline->ptr,str,mess->nchars);
   }

}
```

Figure 4.4 Adding a Line of Text to the Ring Buffer

V. NPS INTERFACE BUILDER OUTPUT

The NPS IB program produces two forms of output. One type of output is a "C" language graphics program skeleton that can be integrated with the user's application program to produce a whole software system. The second type of output is an ASCII file that lists all graphics information saved for a particular interface. The information saved in the ASCII file can be read back into the NPS IB program, thus enabling a user to re-edit a previously constructed interface.

A. SAVING INFORMATION TO OUTPUT FILES

When the user elects the save option from the main menu, the main driver portion of the program calls routine **save-design** to create both output files. The **savedesign** routine is located in the file **save.c.** The **savedesign** routine obtains a user-input file name by calling the keyboard routine for user input, and then allocates memory and opens two files for writing. The user-input file name is appended with a ".c" extension and is given to the file intended for use as a "C" language program. An unaltered version of the user-input file name is given to the ASCII output file. Naming the files in this manner relieves a user from the burden of remembering two totally different file names. Addition of the ".c" extension to the name for the "C"

language output file facilitates the compilation and execution of that file by eliminating the need to copy from a file without a ".c" extension to a file with a ".c" extension.

After opening both files, the routine proceeds to write information to those files. The "C" language output file gets a global include line, routine name, and declaration of variables. The ASCII output file gets comments and coordinates for the size of the design window. The routine then enters four consecutive loops that traverse the linked lists and write the information from the lists to the output file. Sample code for this is shown in Figure 5.1. For the "C" language output file, the information is saved in the form of a call to a routine. When the "C" language output file is linked with support files, this routine will obtain information from the call to create an array entry for a particular control. The support routines will use the graphics information saved in the array to recreate the interface designed in the NPS IB program. The information is transferred from a linked list format to an array format because the array permits faster access of control current values by eliminating the need to traverse lists.

When the linked lists are traversed, graphics information is also saved to the ASCII output file, in the form of tokens followed by individual control values read in

```
/* for the number of toggle boxes we have ... */
for (togboxcurrent = togboxhead; togboxcurrent != NULL;
               togboxcurrent = togboxcurrent->next)
{
   /* increment toggle box count */
   togboxnum++;

   /* write them to the file */
   fprintf(outfile,"\n   maketogbutton(\"%s\",%f,%f,%f,%f,
\n\t\t%f,%f,%d);",
       togboxcurrent->title,togboxcurrent->Xcoord,
togboxcurrent->Ycoord,togboxcurrent->llxbd, togboxcurrent->
       llybd, togboxcurrent->urxbd,togboxcurrent->
       urybd,togboxnum);
  fprintf(outfiletwo,"\/* toggle box information *\/\n");
  fprintf(outfiletwo,"togglebox\n");
  fprintf(outfiletwo,"title %s\n",togboxcurrent->title);
  fprintf(outfiletwo,"controlnum %d\n",togboxcurrent->num);
  fprintf(outfiletwo,"Xcoord %f\n",togboxcurrent->Xcoord);
  fprintf(outfiletwo,"Ycoord %f\n",togboxcurrent->Ycoord);
  fprintf(outfiletwo,"lowerleftxbound %f\n",togboxcurrent->
llxbd);
  fprintf(outfiletwo,"lowerleftybound %f\n",togboxcurrent->
llybd);
  fprintf(outfiletwo,"upperrightxbound %f\n",togboxcurrent->
urxbd);
  fprintf(outfiletwo,"upperrightybound %f\n",togboxcurrent->
urybd);
  fprintf(outfiletwo,"togboxend\n\n");
}
```

Figure 5.1 Code for Writing from Linked Lists to Output Files

from the linked lists. The tokens will be used to identify what the values are when the file is scanned and read back into the NPS IB program.

Saving information to the ASCII output file ends with traversal of the last linked list. The remainder of the **savedesign** routine writes the rest of the graphics program skeleton to the "C" language output file. At the end of the

routine, both the ASCII output file and the "C" language output file are closed.

B. SUPPORT ROUTINES FOR THE "C" LANGUAGE OUTPUT FILE

The purpose of the "C" language output file is to produce a working program with recreation of the interface designed with the NPS IB program. The "C" language output file is not a stand alone file, however. It must be linked with support files and routines in order to operate correctly. The concept of this portion of the program is modeled upon the S & B package program. The user-defined main portion of the S & B package makes calls to routines defined in separate files in order to create its windows of sliders and buttons.

The support files and routines for the NPS IB program "C" language output file are located in the files **interface-global.h**, **interfacepkg.h**, and **interfacepkg.c**. The files **interfaceglobal.h** and **interfacepkg.h** contain global constant and variable definitions. **Interfaceglobal.h** contains mostly constant definitions for color and size of tools. **Inter-facepkg.h** contains global variable definitions for holding the computed size of controls, positions of slider buttons and dial needles, x and y boundaries of control icon drawings, and other variables necessary for operation of the program. The file **interfacepkg.c** contains all the routines needed to support the main portion of the program created by NPS IB. The routine **maketogbutton** reads in graphics

information for toggle boxes and enters each toggle box into a toggle box array. Sample code for this process is shown in Figure 5.2. The routine **makevslider** reads in graphics information about vertical sliders and enters each one into a vertical slider array. **Makehslider** is the routine that enters graphics information about horizontal sliders into a horizontal slider array. **Makedial** takes graphics information about dials and reads that information into a dial array. The routine **initwindow** sets window parameters and attributes, and calls the routine **setcoordi-nates** to compute the values of global coordinate variables for drawing control icons.

```
maketogbutton(title,togx,togy,togllx,toglly,togurx,togury,i-
ndex)
char *title;
float togx,togy,togllx,toglly,togurx,togury;
int  index;

/* adds new button to array of buttons */
{
  buttonTITLE[index] = title;
  togcntrX[index] = togx;
  togcntrY[index] = togy;
  togllxbd[index] = togllx;
  togllybd[index] = toglly;
  togurxbd[index] = togurx;
  togurybd[index] = togury;
  buttonvalue[index] = OFF;
}
```

Figure 5.2 Code for Entering Information into the Toggle Box Array

The routine **drawpicture** is called from the main driver program. Four separate drawing routines, one for each type of control, are called from within **drawpicture.** For-loops internal to each drawing routine control the numbers of each type of control drawn. The for-loop index provides access to the array of information for each control.

Other routines in the package include those that return the current value for each control, and those that cause the controls to operate. The mechanism which runs controls for the "C" language output file program is exactly the same as that described for the NPS IB program itself. The mechanism is detailed in Section I of Chapter III.

C. THE ASCII OUTPUT FILE AND THE RE-EDIT FEATURE

The re-edit feature is selected from the program main menu. The re-edit feature makes the NPS IB program more versatile because it permits a user to read into NPS IB the information for a previously created interface, and allows continuation of editing on that interface. A re-edited interface must be explicitly saved if the user desires to keep it. Changes are not automatically recorded to the output files.

Operation of the re-edit feature is quite straightforward. The NPS IB linked lists are cleared to NULL, essentially wiping any previous interface out of the design window. Then the ASCII file is scanned and information read from the file is used to create new linked lists. When the

new linked list is traversed by the program drawing routine, the new control icons are drawn in the design window.

Routines in two separate files give the NPS IB program its re-edit option. The file **edit.c** contains code to read in a file name, process tokens and values, and build the linked lists. The file **scan.c** contains the routines necessary to scan the ASCII file information.

The format of the ASCII output file is crucial to the correct operation of the re-edit feature. Comments must be bracketed according to the "C" language convention for indicating comments. Tokens must always have the same name, and a token and its associated value must always appear on the same line, separated by at least one space. Each block of information for a control must end with a special token which then signals the editing routine to process the information received. If the ASCII file is edited directly to improper form, or if the user reads in a file not in the ASCII format, an error will occur.

The **readfile** routine in file **edit.c** processes tokens read in from the ASCII output file. The routine asks the scanner to pass it a token number, then switches on the value of the token to determine which global variable applies. When the type of the token is identified, the routine asks the scanner for another token, which it knows is the value assigned to the token type because of the arrangement of the ASCII output file. When the routine

switches on a token type that denotes the end of a block of information for a control, the appropriate processing routine is called to dynamically allocate memory for a record structure and to enter control attributes into the record. Linked list maintenance is also taken care of in the processing routines. These routines are also located in the file **edit.c.**

The scanner routines are located in the file **scanner.c.** The routine **fill_linebuf** performs file input/output by reading from the file into a line buffer. As long as the line buffer is filled with input from the file, the routine **nexttok** can process input. It does this by returning token codes to the routine **gettoken** based on characters seen. A space, for example, is ignored and thus the routine returns a zero to **gettoken.** A letter on the other hand, is a legitimate character, and the code for an identifier is returned to **gettoken** for further processing. In **gettoken,** an identifier token is compared against a list of specially selected tokens. If the identifier token matches a token in the list, the identifier token takes on the value of the token in the list. This value is then returned to the **read-file** routine.

VI. CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

The NPS IB provides graphics students at NPS with a simple interface builder for application programming. It is quick, easy to learn, and easy to use. An application programmer, for example, can use NPS IB to create a graphics program that opens a window and draws three sliders in less than five minutes. Yet, the NPS IB falls far short of goals and expectations envisioned for the original project. Although the basic functionalities of the program are successfully implemented, the program lacks detailed control icon drawings and routines which would give the user more freedom and flexibility to create professional-looking interfaces. An interface builder should be flexible enough to permit a user to accomplish almost any action he can think to perform. The NPS IB fails on this point.

Programming an interface generator is a complex and difficult task. To provide a user with maximum detail and functionality, the complexity of the program seems to increase proportionately. David Tristram of NASA Ames writes that it took him nearly a year and one-half to develop and implement his panel package [Ref. 5]. His result is a slick and professional-looking product with endless possibilities. But even his program has its limitations. It lacks an interactive capability, and

requires a fairly good understanding of the "C" programming language to use. To develop a really useful interface generator for the IRIS workstations, the capabilities of the NPS IB program must eventually be melded with a product such as the NASA Ames Panel Package. Since such a product was the original goal of this study, perhaps the current state of the NPS IB can be viewed as a foundation work towards what might one day be a more powerful and productive interface generator for the Silicon Graphics IRIS workstation.

A. CONCLUSIONS AND LIMITATIONS

1. Controls

The NPS IB limits a user to three choices of controls--a togglebox, a dial, and two versions of a slider. These controls are the most basic ones found in graphics programs, and as such, the selection of controls provided by NPS IB should be adequate for the majority of programmers. Many additional types of controls are possible, however, and could easily be included in the NPS IB selection. A larger inventory of controls would make NPS IB a much more versatile tool.

In addition, NPS IB limits a user to a single icon representation of each control. This is perhaps the single greatest restrictive feature of the NPS IB program. Many versions of toggleboxes, dials, and sliders are possible, through variation of shape and size. The user should

either: (1) be able to pick and chose icons of proper shape and size from an expanded control palette, or (2) be able to alter the shape or size of the control interactively after selecting the control from the palette window. This would allow for greater user creativity in designing interfaces and would ultimately result in greater satisfaction for the user overall.

2. Color

The NPS IB program currently uses a repertoire of six colors to draw window backgrounds and control icons. The user is limited to an interface designed in five colors, unless he edits the color portion of the interface output file directly. This should not be necessary, however. Just as a user should be permitted to select shape and size of controls, he should be allowed to adjust color as well. The NPS IB program should permit the user to utilize any color combinations that the IRIS workstation provides. The NPS IB program does the user a great disservice in preventing him from exercising the color capabilities of the IRIS machine.

Like shapes and size, color can help to distinguish variation among various types of controls. Choice of color scheme for an interface can also be a critical design issue for an application programmer. Lack of color adjustment on part of the NPS IB is another serious deficiency of the program. A mechanism for adjusting color in the NPS IB

would contribute greatly to user creativity and satisfaction.

3. Icon Labelling

The NPS IB program labels each togglebox with its user-input name and each dial and slider with its user-input name and current value. For a togglebox, the name label is aligned to the right of and even with the bottom of the icon itself. For dials and sliders, the name label is centered on the vertical axis of the control and placed directly over top of the icon drawing. The current value of dials and sliders is also centered on the vertical axis of the control but is placed directly beneath the icon drawing. The NPS IB limits the user to these label placements. Provisions for interactive placement of labels and additional text would make the NPS IB program a more flexible tool for designing user interfaces.

B. FUTURE WORK

Future work on the NPS IB should be directed in two basic areas. The first area involves the development of an increased control icon inventory and addition of program routines which would alleviate program limitations and user restrictions discussed in Section A of this chapter. The second area concerns improvements to the user interface of the NPS IB itself.

1. Expanded Inventory of Controls

Work in this area should focus upon building a library of controls, to include such things as palettes, pucks, checkboxes, radio buttons, meters, and strip charts, as well as different varieties of toggleboxes, dials, and sliders. Ideas can be drawn from the user interface features of the Macintosh or NeXT computers, or from SIGRAPH videotapes, or even from David Tristram's Panel Package. The control inventory might also include useful interface features such as a file finder, input boxes for text, and a message buffer similar to the one displayed by NPS IB. The icons representing these functionalities could be interactively placed in the design window by the user, in the same way that controls are placed in the design window now.

2. Adding Controls to NPS Interface Builder

Adding a control to NPS IB is currently a multistep process. The first step involves creating a data record which will hold all the pertinent graphics information about the control. The next step is to draw the control icon in the control palette window. The third step is to make appropriate changes to files in the NPS IB program which will draw the control to the design window (draw.c), save the graphics information for the control (process.c), permit the control to be moved or deleted (move.c), permit the control to be operated (run.c), and which will write the

graphics data for the control to an output file (save.c). Steps one and two are not complicated, but step three involves quite a bit of work. A simpler method of adding controls to NPS IB needs to be devised, especially if the inventory of controls is to be greatly expanded. Perhaps one method of simplifying the process of adding controls to NPS IB would be to restructure the program so that any and all changes would be confined to one file or module. This would save time and effort by reducing the overhead needed to access many different files or modules for change.

3. Interactive Size, Shape, and Color Adjustment

To allow a user greater control over the appearance of an interface created with NPS IB, routines should be added to the program to permit interactive size, shape, and color adjustments to controls. Sizing and shaping mechanisms might be patterned after those used by the Sun workstation Framemaker program or after Apple Macintosh programs like Macdraw. A color adjustment mechanism might employ the IRIS workstation's own cedit program. Any changes to size, shape, or color generated by these routines would then be saved to the data record for that control and subsequently written to the NPS IB output file.

4. Improving Interface of NPS Interface Builder

One oft-repeated rule of interface design is that a program should help prevent a user from committing errors. Unfortunately, this rule is not often followed in the NPS IB

program. Although the message buffer window attempts to assist a user by offering instructions and verification information, many user actions are immediately implemented by the program without seeking confirmation. This flaw in the program could allow a user to mistakenly delete a control, and could force him to repeat the process of selecting, naming, and placing the control over again. Further work in this area would involve the implementation of confirmation windows or messages which would give the user pause to consider his action and would permit him to back out of the action if so desired.

LIST OF REFERENCES

1. Fisher, Alan S., CASE--Using Software Development Tools, John Wiley and Sons, Inc., 1988.

2. NeXT Inc., Online User's Manual for the NeXT Computer, Chapter 7, Fremont, California, 1989.

3. Telephone conversation between Mary Ellen Welch, Oakland Group Inc., and the author, 25 January 1990.

4. Hartson, H. Rex and Hix, Deborah, "Human-Computer Interface Development: Concepts and Systems," ACM Computing Surveys, Association for Computing Machinery, 1989.

5. Tristram, David A., Online Documentation for the Panel Package, NASA Ames Research Center, 1989.

6. Tristram, David A., Panel Library Programmer's Manual, Version 9, NASA Ames Research Center, 1989.

7. Shneiderman, Ben, Designing the User Interface, Addison-Wesley Publishing Company, 1987.

8. Dumas, Joseph S., Designing User Interfaces for Software, Prentice-Hall, Inc., 1988.

9. Silicon Graphics, Inc., IRIS User's Guide, GT Graphics Library User's Guide, Mountain View, California, 1987.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Dr. Michael J. Zyda, Code 52Zk Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	15
4.	LT Susan Lynn Dunlap 16795 Gallop Drive Morgan Hill, California 95037	2
5.	John Maynard Code 402 Naval Ocean Systems Center San Diego, California 92152	1
6.	Duane Gomez Code 433 Naval Ocean Systems Center San Diego, California 92152	1
7.	James R. Louder Naval Underwater Systems Center Combat Control Systems Department Building 1171/1 Newport, Rhode Island 02841	1
8.	Research Administration, Code 012 Naval Postgraduate School Monterey, California 93943-5000	1